Coding K-NN.

Submit your assignments on Gradescope.
Please name your coding assignment as 'HW1.py'.
Use the provided Python template file and complete the functions ONLY (DO NOT edit function definitions, code outside the function, or use other libraries).

This is a coding assignment.

In this assignment, we will implement k-NN.
More specifically, we are interested in seeing the effect of varying **k** on the performance.

The dataset we will use in this assignment is named *iris (in our drive)*(can be downloaded from here)
https://archive.ics.uci.edu/dataset/53/iris

Any submission with an error would result in 0 points. Unless stated otherwise, use the default parameters provided by the libraries when confused.

(1) Fill in the function ***read_data***, which takes in the filename as a string, and returns a pandas dataframe.
Hint: You may find the *read csv* function from the pandas library useful.

(2) Fill in the function ***get_df_shape***, which takes in the pandas dataframe as input and returns the shape as a tuple as output. The shape means the dimensions of the data.
Hint: pandas dataframe instances have a variable shape.

(3) Fill in the function ***extract_features_label*** that returns features and the label from the data. See the function definition (and sample main function) for input/output types. The features we are interested in are 'sepal.length', 'sepal.width', and label 'variety

(4) Fill in the function ***data_split*** that given features and labels split the data into a train/test split (with a given ratio). The function returns 4 numpy arrays in the following order *x_train, y_train, x_test, y_test.*
(x_train, y_train, x_test, y_test = data_split(features, label, 0.33)
Hint: You can use *train test split* from the *sklearn* library, train_test_split

(5) Fill in the function ***knn_test_score*** that takes the train test data as generated by the previous question and applies kNNs on the train data, and returns accuracy on the test data, given the number of neighbors to use as first parameters.
Hint: You can use the *KNeighborsClassifier* function from the *sklearn* library.

(6) Fill in the function ***knn_evaluate_with_neighbours*** that takes the data (same as above) but iterates over (*n_neighbours_min* to *n_neighbours_max*). Note, both min and max are inclusive.
Hint: accu.append()